BookRAG: A Hierarchical Structure-aware Index-based Approach for Retrieval-Augmented Generation on Complex Documents

Shu Wang
The Chinese University of Hong Kong, Shenzhen
shuwang3@link.cuhk.edu.cn

Yingli Zhou
The Chinese University of Hong Kong, Shenzhen
yinglizhou@link.cuhk.edu.cn

Yixiang Fang*
The Chinese University of Hong Kong, Shenzhen
fangyixiang@cuhk.edu.cn

ABSTRACT

As an effective method to boost the performance of Large Language Models (LLMs) on the question answering (QA) task, Retrieval-Augmented Generation (RAG), which queries highly relevant information from external complex documents, has attracted tremendous attention from both industry and academia. Existing RAG approaches often focus on general documents, and they overlook the fact that many real-world documents (such as books, booklets, handbooks, etc.) have a hierarchical structure, which organizes their content from different granularity levels, leading to poor performance for the QA task. To address these limitations, we introduce BookRAG, a novel RAG approach targeted for documents with a hierarchical structure, which exploits logical hierarchies and traces entity relations to query the highly relevant information. Specifically, we build a novel index structure, called BookIndex, by extracting a hierarchical tree from the document, which serves as the role of its table of contents, using a graph to capture the intricate relationships between entities, and mapping entities to tree nodes. Leveraging the BookIndex, we then propose an agent-based query method inspired by the Information Foraging Theory, which dynamically classifies queries and employs a tailored retrieval workflow. Extensive experiments on three widely adopted benchmarks demonstrate that BookRAG achieves state-of-the-art performance, significantly outperforming baselines in both retrieval recall and QA accuracy while maintaining competitive efficiency.

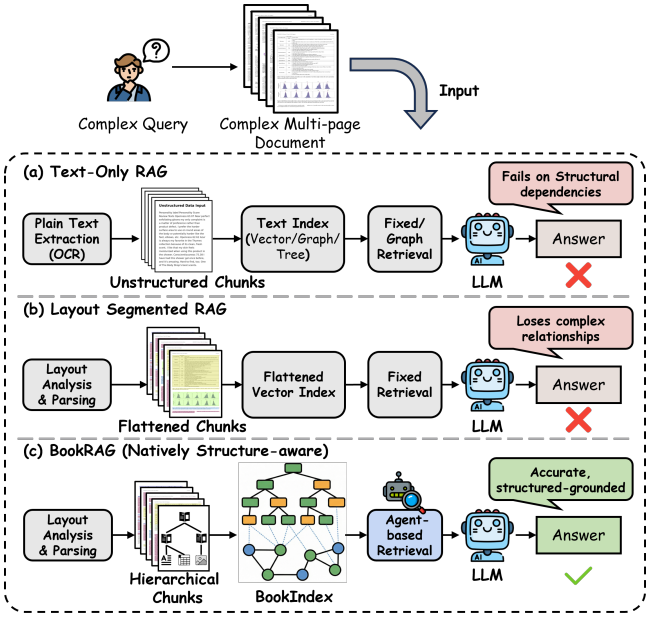


Figure 1: Comparison of existing methods and BookRAG for complex document QA.

PVLDB Reference Format:
Shu Wang, Yingli Zhou, and Yixiang Fang. BookRAG: A Hierarchical Structure-aware Index-based Approach for Retrieval-Augmented Generation on Complex Documents. PVLDB, 19(9): 2358 - 2371, 2026.
doi:10.14778/3819518.3819556

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/sam234990/BookRAG.

1 INTRODUCTION

Large Language Models (LLMs) such as Qwen 3 [62] and Gemini 2.5 [13] have revolutionized Question Answering (QA) [15, 64, 70]. The industry has increasingly adopted LLMs to build QA systems that assist users and reduce manual effort in many applications [70, 72], such as financial auditing [30, 38], legal compliance [7], and scientific discovery [56]. However, directly relying on LLMs may lead to missing domain knowledge and generating outdated or unsupported information. To address these issues, Retrieval-Augmented Generation (RAG) has been widely adopted [18, 23] by retrieving relevant domain knowledge from external sources and using it to guide the LLM during response generation. On the other hand, in real-world enterprise scenarios, domain knowledge is often stored in long-form documents, such as technical handbooks, API reference manuals, and operational guidebooks [48]. A notable feature of such documents is that they follow a book-like structure, characterized by intricate layouts and rigorous logical hierarchies (e.g., explicit tables of contents, nested chapters, and multi-level sections). In this paper, we aim to design an effective RAG system for QA over long and highly structured documents.

• **Prior works.** The existing RAG approaches for document-level QA generally fall into two paradigms, as illustrated in Figure 1. The first paradigm relies on OCR (Optical Character Recognition) to convert the document into plain text, after which any text-based RAG method can be directly applied. Among text-based RAG methods, state-of-the-art approaches increasingly adopt *graph-based RAG* [5, 67, 71], where graph data serves as an external knowledge source because it captures rich semantic information and the

*Yixiang Fang is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 19, No. 9 ISSN 2150-8097.
doi:10.14778/3819518.3819556

Table 1: Comparison of representative methods and our BookRAG.

Type	Representative Method	Core Feature	Multi-hop Reasoning	Document Parsing	Query Workflow
Graph-based	RAPTOR [45]	Recursive summarization	✓	✗	Static
	GraphRAG [17]	Global community detection	✓	✗	Static
Layout segmented	MM-Vanilla	Multimodal retrieval	✗	✓	Static
	SuperRAG [63]	Spatial layout graph	✗	✓	Static
	DocETL [46]	LLM-based document processing pipeline	✗	✓	Manual
Doc-Native	**BookRAG (Ours)**	Structure-aware Index & Agent-based retrieval	✓	✓	Dynamic

relational structure between entities. As shown in Table 1, two representative methods are GraphRAG [17] and RAPTOR [45]. Specifically, GraphRAG first constructs a knowledge graph (KG) from the textual corpus, and then applies the Leiden community detection algorithm [51] to obtain hierarchical clusters. Summaries are generated for each community, providing a comprehensive, global overview of the entire corpus. RAPTOR builds a recursive tree structure by iteratively clustering document chunks and summarizing them at each level, enabling the model to capture both fine-grained and high-level semantic information across the corpus.

In contrast, the second paradigm, *layout-aware segmentation* [4, 52], parses the document into structured blocks (e.g., paragraphs, tables, images) to preserve the original layout and avoid fragmentation caused by fixed-size chunking. These blocks often exhibit multimodal characteristics, and a typical approach is to apply multimodal retrieval to obtain relevant content. Besides, SuperRAG [63] organizes layout units into a graph based on spatial proximity and reading order to expand the retrieval context. Another state-of-the-art approach, DocETL [46], provides a declarative interface that requires users to manually define LLM-based processing pipelines to analyze retrieved blocks through task-specific operations.

• **Limitations of existing works.** However, these methods suffer from two fundamental limitations (**L** for short): **L1: Failure to capture the deep connection of document structure and semantics.** Text-based approaches cannot capture the structural layout of the document, resulting in the loss of important relationships stored in the hierarchical blocks, such as tables nested within a specific section. While layout segmented methods preserve document structure, they do not explicitly exploit the logical relationships between discrete blocks. This limits their capacity for multi-hop reasoning and makes it difficult to aggregate evidence scattered throughout the document, a fundamental requirement for addressing complex, real-world queries. **L2: Static query workflows.** In real-world QA scenarios, user queries are highly heterogeneous, ranging from simple keyword lookups to complex multi-hop questions that require synthesizing evidence scattered across different parts of the document. Applying a uniform strategy, such as static or manually predefined workflows, to diverse needs is inefficient; for example, complex queries often require question decomposition, whereas simple queries do not.

• **Our technical contributions.** To bridge this gap, we introduce **BookRAG**, the first retrieval-augmented generation method built upon a document-native **BookIndex**, designed for document QA tasks. Specifically, to capture the deep connection among relations in the document, BookIndex organizes information through two complementary structures. First, to preserve the document's native logical hierarchy, we organize the parsed content blocks into a hierarchical tree structure, which serves as its table of contents. Second, to capture the intricate relations within these blocks, we construct a KG containing fine-grained entities. Finally, we unify these two structures by mapping the KG entities to their corresponding tree nodes.

However, effective multi-hop reasoning on the graph relies on a high-quality KG [67, 71], which is often compromised by entity ambiguity (e.g., distinct entities with names like "LLM" and "Large Language Model"). To address this, we propose a novel gradient-based entity resolution method that analyzes the similarity distribution of candidate entities. By identifying sharp drops in similarity scores, we can efficiently distinguish and merge coreferent entities, thereby ensuring graph connectivity and enhancing reasoning capabilities.

Building upon the BookIndex, we address the static query workflows (**L2**) by implementing an **agent-based retrieval**. Specifically, BookRAG first classifies user queries based on their intent and complexity, and then dynamically generates tailored retrieval workflows. Grounded in *Information Foraging Theory* [42], our retrieval process mimics foraging by using *Selector* to narrow down the search space via information scents and *Reasoner* to locate highly relevant evidence.

We conduct extensive experiments on three widely adopted datasets to validate the effectiveness and efficiency of our BookRAG, comparing it against several state-of-the-art baselines. The experimental results demonstrate that BookRAG consistently achieves superior performance in both retrieval recall and QA accuracy across all datasets. Furthermore, our detailed analysis validates the critical contributions of our key features, such as the high-quality KG and the agent-based retrieval mechanism.

We summarize our contributions as:

- We introduce **BookRAG**, a novel method that constructs a document-native *BookIndex* by integrating a hierarchical tree of document layout blocks with a KG storing fine-grained entity relations.
- We propose an **Agent-based Retrieval** approach inspired by Information Foraging Theory, which dynamically classifies queries and configures optimal retrieval workflows to locate highly relevant evidence within documents.
- Extensive experiments on multiple benchmarks show that *BookRAG* significantly outperforms baselines, achieving state-of-the-art performance in solving complex document QA tasks while maintaining competitive efficiency.

Outline. We review related work in Section 2. Section 3 introduces the problem formulation, IFT, and RAG workflow. In Section 4,

we present the structure of our BookIndex and its construction. Section 5 presents our agent-based retrieval, elaborating on the query classification and operators used in the structured execution of BookRAG. We present the experimental results and detailed analysis in Section 6, and conclude the paper in Section 7.

2 RELATED WORK

In this section, we review the related work, including LLMs in document analysis and the modern representative RAG approaches.

(1) LLMs in document analysis. Recent advances in LLMs have offered opportunities to leverage LLMs in document analysis. Due to the robust semantic reasoning capabilities of LLMs, an increasing number of works focus on transferring unstructured documents (e.g., HTML, PDFs, and raw text) into structured formats, such as relational tables [1, 6, 26, 39]. For example, Evaporate [1] utilizes LLMs to synthesize extraction code, enabling cost-effective conversion of semi-structured web documents into structured databases without heavy manual annotation. In addition, several LLM-based document analysis systems have been proposed to equip standard data pipelines with semantic understanding [29, 40, 46, 53]. For instance, LOTUS [40] extends the relational model with semantic operators, allowing users to execute SQL-like queries with LLM-powered predicates (e.g., filter, join) over unstructured text corpora. Similarly, DocETL [46] introduces an agentic framework to optimize complex information extraction tasks. Furthermore, another line of research proposes to directly analyze or parse documents by viewing the document pages as images, thereby preserving critical layout and visual information [27, 33, 54]. In contrast to these systems that focus on structuring data for database operations or general-purpose extraction, BookRAG is specifically designed for complex document QA. It distinguishes itself by preserving the document's native logical hierarchy and multi-level sections to build a structure-aware index. Crucially, our approach captures intricate relations within the document's layout and content, going beyond simple structured table extraction or general data cleaning to support deep reasoning over complex document topologies.

(2) RAG approaches. RAG has been proven to excel in many tasks, including open-ended question answering [25, 47], programming context [9, 10], SQL rewrite [31, 50], and data cleaning [36, 37, 43]. The naive RAG technique relies on retrieving query-relevant contexts from external knowledge bases to mitigate LLMs' "hallucination." Recently, many RAG approaches [17, 19, 20, 22, 28, 34, 45, 49, 55, 58–60, 65, 71] have adopted graph structures to organize the information and relationships within documents, achieving improved overall retrieval performance. For more details, please refer to the recent survey of graph-based RAG methods [41]. Moreover, the Agentic RAG paradigm has been widely studied, employing autonomous agents to dynamically orchestrate and refine the RAG pipeline, thus significantly boosting the reasoning robustness and generation fidelity [2, 24, 61]. Furthermore, layout-aware frameworks such as SuperRAG [63] have been proposed to organize document units into graphs based on their spatial proximity and reading order. Compared to these methods, BookRAG utilizes a structure-aware index that synergizes document hierarchies with fine-grained entity graphs. This enables precise retrieval at the **block level**, surpassing the accuracy of traditional fixed-size chunking or page-level approaches. Moreover, BookRAG adopts a **dynamic workflow** grounded in Information Foraging Theory to navigate intricate relations more efficiently than standard predefined or data-driven pipelines.

3 PRELIMINARIES

This section formalizes the problem setting of complex document QA, introduces the foundational Information Foraging Theory (IFT), and briefly reviews the general workflow of RAG systems.

3.1 Problem Formulation

We study the problem of Question Answering (QA) over complex documents, which aims to answer user queries based on long-form documents [4, 11, 35]. Formally, a document D is represented as a sequence of N pages, $D = \{P_i\}_{i=1}^{N}$. These pages collectively contain a sequence of content blocks $\mathcal{B} = \{b_j\}_{j=1}^{M}$, where each block b_j represents a distinct element (e.g., text segment, section header, table, or image) organized within a logical chapter hierarchy. Given a user query q, the goal is to generate an accurate answer A, ideally grounded in a specific set of evidence blocks $E \subset \mathcal{B}$. The task is formulated as developing a method $\mathcal{S}$ that maps the structured document and the query to the final answer:

$$A = \mathcal{S}(D, q) \tag{1}$$

where $\mathcal{S}$ should navigate both the sequential page content and the logical hierarchy of D to synthesize the response. Specifically, BookRAG retrieves information at the **content block** level. Each extracted block is represented as a node within the **BookIndex**, and BookRAG generates answers based on these fine-grained elements to enable precise evidence localization.

3.2 Information Foraging Theory

Information Foraging Theory (IFT) [42] provides a framework for understanding information access as a process analogous to animal foraging. It suggests that users follow cues, known as **information scent** (e.g., keywords or icons), to navigate between clusters of content, known as **information patches** (e.g., sections in handbooks). The goal is to maximize the rate of valuable information gain while minimizing effort, guiding the decision to either stay within a patch or seek a new one.

Consider experts seeking a solution to a specific problem within a large technical handbook. They first extract key terms related to the problem, which act as information scent. This scent guides them to navigate towards one or more promising sections (the information patches). Within these patches, they analyze the diverse content to extract the precise knowledge required to formulate a final answer.

3.3 RAG workflow

Retrieval-Augmented Generation (RAG) systems typically operate in a two-phase framework [5, 17, 41]. In the **Offline Indexing** phase, unstructured corpus data is organized into a structured index, which can take various forms such as vector databases or KGs [71]. Subsequently, in the **Online Retrieval** phase, the system retrieves relevant components (e.g., text chunks or subgraphs) based on the user query q to inform the LLM's generation. However, these general workflows often treat the index as a structure derived purely from content, potentially detaching it from the document's original logical hierarchy. In contrast, our approach seeks to deeply integrate these retrieval structures with the document's native tree topology.

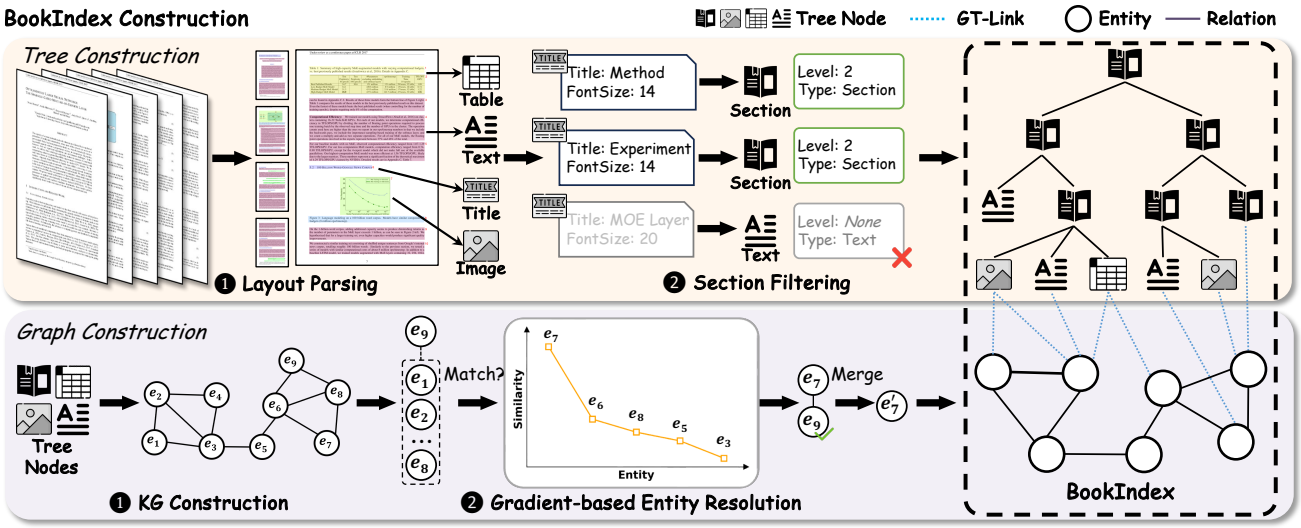


Figure 2: The BookIndex Construction process. This phase includes Tree Construction, derived from Layout Parsing and Section Filtering, and Graph Construction, which involves KG Construction and Gradient-based Entity Resolution.

4 BOOKINDEX

This section introduces our proposed **BookIndex**, a hierarchical structure-aware index designed to capture both the explicit logical hierarchy and the intricate entity relations within complex documents. We first formally define the structure of the BookIndex (B). Subsequently, we elaborate on the sequential, two-stage construction process: (1) **Tree Construction**, which parses the document's layout to establish a hierarchy of nodes, each categorized by type; and (2) **Graph Construction**, which extracts fine-grained entity knowledge from the tree nodes and refines it through a novel gradient-based entity resolution method.

4.1 Overview of BookIndex

We formally define our **BookIndex** as a triplet $B = (T, G, M)$. Here, $T = (N, E_T)$ represents a *Tree* structure where N is the set of nodes derived from the document's explicit logical hierarchy (each corresponding to an extracted primitive block, e.g., text, tables, images), and E_T denotes their nesting relationships. $G = (V, E_G)$ is a *Knowledge Graph* that captures fine-grained entities (V) and their relations (E_G) scattered throughout the document. Finally, $M : V \rightarrow \mathcal{P}(N)$ is the *Graph-Tree Link (GT-Link)*, which links each entity in V to the set of specific tree nodes in N from which it was extracted. These links are crucial for capturing the intricate, cross-sectional relations within the document. The hierarchical tree nodes in T serve as the document's native *information patches*, providing structured contexts for information seeking. Meanwhile, the entities and relations in G, connected via M, act as the rich *information scent* that guides navigation between and within these patches.

Figure 2 provides an example of our BookIndex. The Tree component, positioned at the top, organizes the document into a hierarchical structure, where content blocks such as text, tables, and images serve as leaf nodes nested within section nodes. The Graph component is composed of entities and relations extracted from these nodes. The GT-Link, illustrated by the blue dotted lines, explicitly connects these entities back to their corresponding tree nodes, thereby grounding the semantic entities within the document's logical hierarchy.

4.2 Tree Construction

The first stage transforms the raw document into a structured hierarchical tree T. This involves two key steps: layout parsing and section filtering.

4.2.1 Layout Parsing. The Layout Parsing phase transforms document pages D into a sequence of multimodal blocks $\mathcal{B}$ by performing high-fidelity element extraction and content recognition. While this phase is modular and compatible with various layout tools, we use the MinerU [52], which employs decoupled model backbones (e.g., NaViT [16] and Qwen2-Instruct [62]) for global layout analysis and targeted content recognition. Specifically, the pipeline processes each page to extract diverse layout regions and recognizes their contents, converting them into digitized formats such as text, formulas, and structured tables. The output is a sequence of blocks, $\mathcal{B} = \{b_1, b_2, \cdots, b_k\}$, arranged according to the original reading order of the document. Each block $b_i = (c_i, \tau_i, f_i)$ is defined as a triplet: c_i is the raw content (e.g., text, formulas, or structured tables), τ_i is the layout type (e.g., `Title, Text, Table, Image`), and f_i is a vector of associated layout features (e.g., "FontSize", bounding box).

4.2.2 Section Filtering. While layout parsing can classify some blocks as `Title` type, it does not assign hierarchical levels and often misclassifies elements such as image captions and borderless table headers. To address this, we prompt an LLM with examples to identify heading levels, establish parent-child relationships, and detect incorrectly parsed section blocks. We first select the candidate subset $\mathcal{B}_{\text{title}} \subset \mathcal{B}$ (where $\tau_i =$ `Title`) and perform the analysis in batches to ensure scalability for long documents. For each candidate, the LLM synthesizes its semantic text content (e.g., heading-like phrasing such as "Chapter 1"), visual layout features (e.g., font size and height), and page indices (ensuring logical continuity across the document). Based on these features, the LLM determines whether

the block is a valid title or a parsing error, and for valid titles, identifies their correct parent sections and determines their hierarchical levels $l_i \in \{1, 2, \cdots\}$. Blocks identified as parsing errors are corrected back to regular `Text` blocks. This step yields a refined hierarchical structure that serves as the backbone of our *Tree* structure.

Finally, the tree $T = (N, E_T)$ is constructed using the refined hierarchical backbone. The node set N comprises all blocks after the above process, where each node $n \in N$ retains its content c_i and corrected type τ_i' (e.g., `Section`, `Text`, `Table`, and `Image`). To establish the edge set E_T, we leverage the fact that the original sequence $\mathcal{B}$ follows the document's natural reading order, which essentially represents a pre-order traversal of the logical hierarchy. By sequentially traversing this sequence, each node is attached as a child to the most recent preceding `Section` node of the appropriate hierarchical level l_i. This process transforms the flat sequence into a complete tree structure that accurately preserves both the document's physical continuity and its logical depth.

As shown in Figure 2, the *Layout Parsing* phase identifies diverse blocks, typing them as `Title`, `Text`, `Table`, and `Image`. During the *Section Filtering* phase, the `Title` candidates (e.g., "Method", "Experiment", and "MOE Layer") are analyzed by the LLM. The blocks "Method" and "Experiment" (both with "FontSize: 14") are correctly identified as `Section` nodes at "Level: 2". Conversely, the "MOE Layer" block ("FontSize: 20"), which was erroneously tagged as `Title` by the parser, is re-classified by the LLM as a `Text` node with "Level: None". Following the assignment of hierarchical levels to the identified `Section` nodes, we sequentially traverse the original block sequence to assemble the final 4-layer tree structure. This process ensures that all content nodes are correctly attached under their respective section backbones, preserving the document's complete logical depth.

4.3 Graph Construction

Once the tree T is established, we proceed to populate the Knowledge Graph G by extracting and refining entities from the tree nodes.

4.3.1 KG Construction. We iterate over each node $n_i \in N$ from the previously constructed tree T. For each node n_i, we extract a subgraph $g_i = (V_i, E_{Ri})$ based on its content c_i and final node type τ_i'. This extraction is modality-dependent: if the node is text-only, an LLM is prompted to extract entities and relations, while for nodes containing visual elements (e.g., $\tau_i' =$ `Image`), a Vision Language Model (VLM) is employed to extract visual knowledge. Crucially, for every entity $v \in V_i$ extracted, its origin tree node n_i is recorded, which is vital for constructing the final mapping M.

Furthermore, to preserve structural semantics for specific logical types (e.g., `Table`, `Formula`), our process first creates a distinct, typed entity (e.g., v_{table} representing the table itself). The other extracted entities from the specific node's content are linked to this primary vertex. For `Table` nodes specifically, row and column headers are also explicitly extracted as distinct entities and linked to v_{table} via a "ContainedIn" relationship.

4.3.2 Gradient-based Entity Resolution. As shown in the literature [67, 71], a well-constructed KG is essential for document question answering. A common challenge in the extraction process is that the same conceptual entity is often fragmented into multiple distinct entities due to abbreviations, co-references, or its varied occurrences across different document sections. This necessitates a robust Entity Resolution (ER) process, which identifies and merges these fragmented entities to refine the raw KG.

Algorithm 1: Gradient-based entity resolution

Input: KG G, New entity v_n, Rerank model $\mathcal{R}$, Entity vector database DB, Vector-search parameter top_k, Gradient threshold g

```
// Vector Search top_k relevant entities in DB.
E_c ← Search(DB, v_n, top_k);
S ← R(E_c, v_n);
// Sort all candidate entities by rerank scores.
Sort(E_c, S);
score ← S[0], Sel ← E_c[0];
// Gradient select similar entities.
for each remaining entity v_c ∈ E_c \ {E_c[0]} do
    if S[v_c] > score * g then
        Sel ← Sel ∪ {v_c}, score ← S[v_c];
    else break;
// Merge entity or add new entity.
if length(Sel) = length(E_c) then
    G ← AddNewEntity(G, v_n), DB ←AddNew(DB, v_n);
else
    if length(Sel) = 1 then v_sel ← Sel[0];
    else v_sel ← LLMSelect(Sel);
    G ← MergeEntity(G, v_n, v_sel), DB ←Update(DB, v_sel, v_n);
return G, DB;
```

However, conventional ER methods are computationally expensive. They are often designed for batch processing across multiple data sources (commonly referred to as dirty ER), aiming to ensure accurate entity resolution by finding all possible matching pairs [12]. This process typically requires finding the transitive closure of all detected matches. That is, to definitively merge multiple entities (e.g., A, B, and C) as the same concept, the system must ideally compare all possible pairs ("A-B", "A-C", and "B-C") to confirm their equivalence. This can lead to a quadratic ($O(n^2)$) number of pairwise comparisons, a process that becomes prohibitively slow and computationally expensive when relying on LLMs for high-accuracy judgments.

To address this, we employ a gradient-based ER method, operating on a single document (simplified as the clean ER), which performs ER incrementally as each new entity v_n is extracted. This transforms the quadratic batch problem into a simpler, repeated lookup task: determining where the single new entity v_n fits among the already-processed entities in the database. This incremental process yields two distinct, observable scoring patterns when v_n is reranked against its top_k most relevant candidates:

- *Case A: New Entity.* If v_n is a new conceptual entity, its relevance scores against all existing entities will be uniformly low, showing no significant gradient or discriminative pattern.
- *Case B: Existing Entity.* If v_n is an alias of an existing entity, its scores will show high relevance to the true match (or a small set of equivalent aliases). Due to the reranker's inherent discriminative limitations, this initial high-relevance set might occasionally contain multiple similar entities. This

high-relevance set is then typically followed by a **sharp decline** (a large "gradient") before transitioning to a **gradual slope** of irrelevant entities.

Our Gradient-based ER algorithm is designed precisely to detect this sharp decline (characteristic of Case B), allowing us to efficiently isolate the high-relevance set. Subsequently, an LLM is utilized for finer-grained distinction when multiple similar entities are identified within this set, differentiating it from the "no gradient" scenario (Case A) without quadratic comparisons.

Algorithm 1 shows the above entity resolution process. For a new entity v_n, we first retrieve its top_k candidates E_c from the vector database DB, which are then reranked by $\mathcal{R}$ against v_n and sorted based on their scores $\mathcal{S}$ (Lines 1-3). We initialize the selection set Sel with the top-scoring candidate $E_c[0]$ and set the initial `score` to $\mathcal{S}[0]$ (Line 4). We then iterate through the remaining sorted candidates (Lines 5-8). The core logic checks if the current score $\mathcal{S}[v_c]$ remains above the previous score scaled by the gradient threshold g (i.e., $\mathcal{S}[v_c] > \texttt{score} * g$). If the score drop is gentle (passes the check), the candidate v_c is added to Sel, and `score` is updated (Lines 7-8); otherwise, the loop breaks (Line 8) as soon as a sharp score drop is detected. Finally, the algorithm makes its decision (Lines 9-14). If the selection set Sel is identical to E_c, this indicates that all candidates passed the gradient check. This corresponds to *Case A*, where the scores lacked discriminative power (i.e., v_n is equally dissimilar to all candidates), so v_n is added as a new entity (Line 9-10). Conversely, if a gradient was found (i.e., $length(Sel) < length(E_c)$), this signals *Case B*. We then select the canonical entity v_{sel} from Sel, using an LLM (Line 13) if the reranker identifies multiple aliases, and merge v_n with it (Lines 12-14). The updated G and DB are then returned (Line 15).

For instance, considering the example in Figure 2, when the new entity e_9 is processed, it is first compared with existing entities in the KG. As depicted in the similarity curve (orange line), e_9 shows high similarity with e_7, followed by a sharp decline in similarity with other entities like e_6, e_8, and e_5. Our gradient-based selection process identifies e_7 as the unique, high-confidence match for e_9. Consequently, e_9 is merged with e_7, enriching the KG with consolidated information as shown in the final merged entity e_7'.

Graph-Tree Link (GT-Link). The GT-Link M is formalized to complete the BookIndex $B = (T, G, M)$. As described in the *KG Construction* phase, the origin tree node n_i is recorded for every newly extracted entity v_i. GT-Link is then refined during entity resolution: when an entity v_n is merged into a canonical entity v_{sel}, the origin node set of v_{sel} is updated to include all origin nodes previously associated with v_n. This aggregation process creates the final mapping $M : V \to \mathcal{P}(N)$, which bi-directionally links the entities in G to the set of their structural locations (nodes) in T.

5 AGENT-BASED RETRIEVAL

Real-world document queries are often complex, necessitating operations like modality-specific filtering, semantic selection, and multi-hop reasoning. To address this, we propose an agent-based approach in BookRAG, which intelligently plans and executes operations on the BookIndex. We first introduce the overall workflow and present two core mechanisms: **Agent-based Planning**, which formulates the strategy, and the **Structured Execution**, which includes the retrieval process under the principles of IFT and generation.

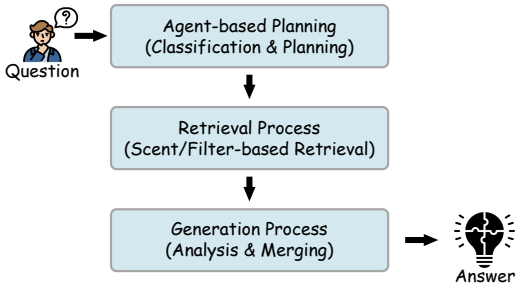


Figure 3: The general workflow of agent-based retrieval in BookRAG, which contains agent-based planning, retrieval, and generation processes.

5.1 Overall Workflow

The overall workflow of agent-based retrieval, illustrated in Figure 3, follows a three-stage pipeline designed to address users' queries systematically.

1. Agent-based Planning. BookRAG first performs *Classification & Planning*. This stage aims to distinguish different query types, such as simple keyword-based queries, multi-hop reasoning queries, and global aggregation queries, since they require different retrieval workflows. For instance, a query like "How do SQL-based and graph-based approaches differ in relation handling?" requires retrieving evidence for both sides and then performing cross-section comparison, rather than directly matching a single passage. Therefore, the planning stage first performs **query classification**. Based on the identified query type and a predefined set of operators designed for the BookIndex, it generates a specific **operator plan** that guides the retrieval and generation strategies. This plan acts as a strategic navigation path, mapping heterogeneous queries to the most effective foraging sequences within the complex document topology.

2. Retrieval Process. Guided by the operator plan, the retrieval process executes *Scent/Filter-based Retrieval*. This stage navigates the BookIndex $B = (T, G, M)$, either utilizing a **scent-based retrieval principle** (e.g., following relevant entities in G) to find information, or employing various filters (e.g., modality type) to refine the selection. Rather than relying on a monolithic retrieval model, this process employs a "narrow-then-reason" strategy through modular operators. This decoupling ensures that each operator focuses on its specialized function, effectively mitigating information noise and retrieving highly relevant information. After reasoning, BookRAG retrieves a set of highly relevant nodes from the BookIndex.

3. Generation Process. Finally, all retrieved information enters the generation stage for *Analysis & Merging*. This stage synthesizes these (often scattered) pieces of evidence, performs final analysis, and formulates a coherent response.

5.2 Agent-based Planning

The planning stage is the core of BookRAG, designed to intelligently navigate our BookIndex $B = (T, G, M)$. To support flexible retrieval, we define four types of operators: Formulator, Selector, Reasoner, and Synthesizer. These operators can be arbitrarily combined to form tailored execution pipelines, each with adjustable parameters. BookRAG dynamically configures and assembles these operators to adapt to the specific requirements of different query categories.

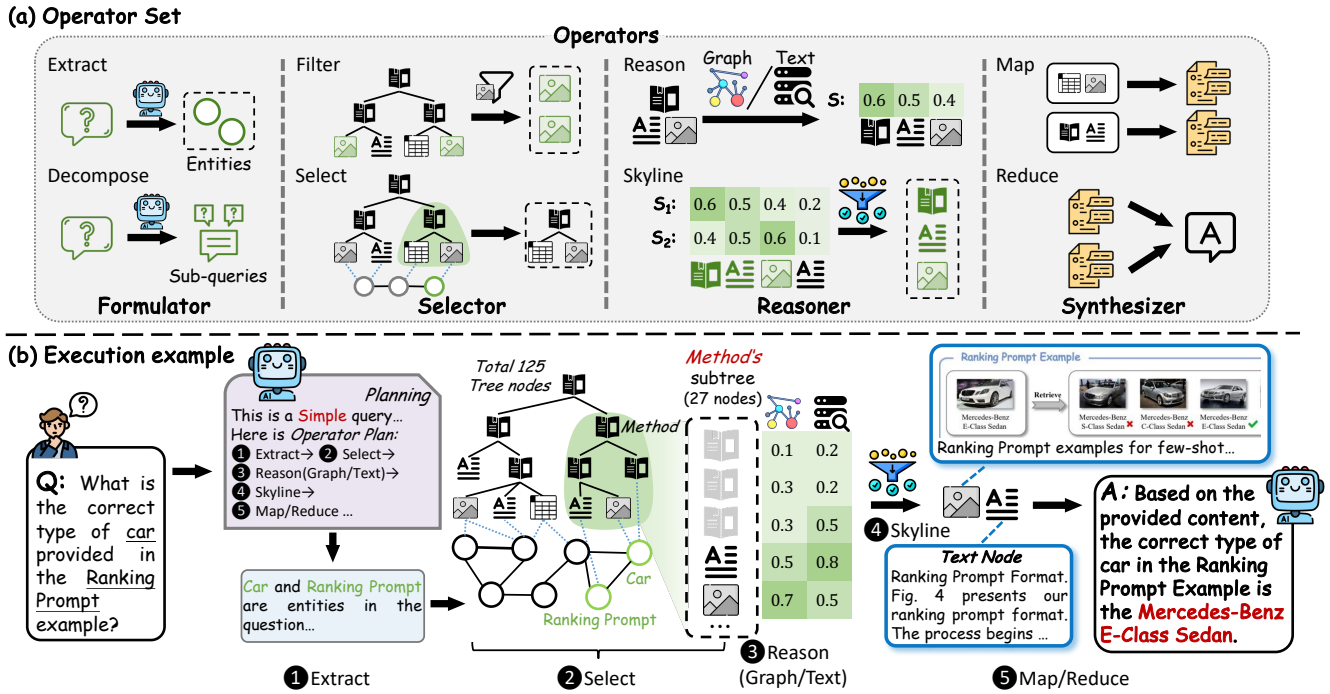


Figure 4: The BookRAG Operator Library and an Execution Example from the MMLongBench dataset: (a) a visual depiction of the four operator types (Formulator, Selector, Reasoner, and Synthesizer) and (b) an execution trace for a "Single-hop" query. The trace demonstrates how the agent plans a sequence to navigate the BookIndex, pruning the search space (from 125 to 27 nodes) and filtering multimodal evidence.

This process involves two sequential steps: first, the agent performs *Query Classification* to determine the appropriate solution strategy, then generates a specific *Operator Plan*.

• **Query Classification**. To enable agent strategy selection, we focus on three representative query categories defined by their intrinsic complexity and operational demands (Table 2): *Single-hop*, *Multi-hop*, and *Global Aggregation*. This classification is crucial because each category requires a different solution strategy. For instance, a *Single-hop* query typically requires a single piece of information retrieved via a *Scent-based Retrieval* operation. In contrast, a *Global Aggregation* query often necessitates analyzing content under multiple filtering conditions, usually involving a sequence of *Filter & Aggregation* operations across various parts of the document. Furthermore, BookRAG is designed to be extensible, allowing for the resolution of a broader range of query types by integrating additional operators.

• **BookIndex Operators**. To execute the strategies identified by classification, we designed a set of operators (O) tailored for the BookIndex $B = (T, G, M)$. These operators, visually depicted in Figure 4(a), define the set of operations the agent can employ for diverse query categories. We also provide a detailed comparison of these operators in the appendix of our technical report [57]. We group them into four types, which we describe in sequence:

❶ *Formulator.* These are LLM-based operators that prepare the query for execution. This category includes `Decompose`, which breaks a complex query into a set of simpler, actionable sub-queries Q_s. It also includes `Extract`, which employs an LLM to identify key entities E_q from the query text and link them to corresponding entities in the KG, G:

$$Q_s = \text{LLM}(P_{Dec}, q) = \{q_1, q_2, \ldots, q_k\} \quad (2)$$

$$E_q = \text{LLM}(P_{Ext}, q) = \{e_1, e_2, \ldots, e_m\} \quad (3)$$

Table 2: Three common query categories addressed in BookRAG.

Query Category	Description	Core Task	Example Query
Single-hop	Queries with a single, distinct information target.	**Scent-based Retrieval**: Retrieve content related to a specific entity or section.	*What is the definition of Information Scent?*
Multi-hop	Queries that require synthesizing information from multiple blocks, often by decomposing into sub-problems.	**Decomposing & Merging**: Decompose into sub-problems, retrieve for each, and synthesize the final answer.	*How do SQL-based and graph-based approaches differ in relation handling?*
Global Aggregation	Queries that require filtering across the entire document and performing calculations.	**Filter & Aggregation**: Apply filters across the document & perform aggregation operations (e.g., Count, Sum).	*How many figures related to IFT are in Section 4?*

Here, q is the original user query, while P_{Dec} and P_{Ext} represent the prompts used to guide the LLM in the decomposition and extraction tasks, respectively.

❷ *Selector.* These operators filter or select specific content ranges from the BookIndex. `Filter_Modal` and `Filter_Range` directly apply the explicit constraints C (e.g., modal types, page ranges) generated during the plan. Operating on the *Tree* $T = (N, E_T)$, these operators produce a filtered subset N_f where the predicate $C(n)$ holds true for each node:

$$N_f = \{n \in N \mid C(n)\} \tag{4}$$

In contrast, `Select_by_Entity` and `Select_by_Section` target contiguous document segments by retrieving subtrees rooted at specific section nodes. This process first identifies a set of target section nodes $S_{\text{target}} \subset N$ at a specified depth, where S_{target} consists of sections either linked to entities E_q via the *GT-Link* M or selected by the LLM. It then retrieves all descendants of these targets to form the selected node set N_s:

$$N_s = \bigcup_{s \in S_{\text{target}}} \text{Subtree}(s) \tag{5}$$

❸ *Reasoner.* These operators analyze and refine selected tree nodes. `Graph_Reasoning` performs multi-hop inference on a subgraph $G'(V', E')$ (extracted from selected nodes N_s) starting from entity e. Starting from the retrieved entities, it computes an entity importance vector $I_G \in \mathbb{R}^{|V'|}$ over the subgraph G' using the PageRank algorithm [21]. These entity scores are then mapped to the tree nodes via the GT-Link matrix M to derive the final tree-node importance score vector $S_G \in \mathbb{R}^{|N_s|}$:

$$I_G = \text{PageRank}(G', e) \tag{6}$$

$$S_G = I_G \times M \tag{7}$$

`Text_Reasoning` evaluates the semantic relevance of the tree node's content to the query q, assigning a relevance score S_T to each node. `Skyline_Ranker` employs the Skyline operator to filter nodes based on these multiple criteria (e.g., S_G and S_T), retaining only those nodes that are not dominated by any others in terms of the specified scoring dimensions.

❹ *Synthesizer.* These operators are responsible for content generation. `Map` performs analysis on specific retrieved information segments to generate partial responses. `Reduce` synthesizes a final coherent answer by aggregating information from multiple sources, such as partial answers or a collection of retrieved evidence.

• **Operator Plan**. After classifying the query (q) into its category (c), the agent's final task is to generate an executable plan P. This plan is a specific sequence of operators $\langle o_1, \ldots, o_n \rangle$ selected from our library O with parameters dynamically instantiated based on q. This process is formulated as:

$$P = \text{Agent}_{\text{Plan}}(q, c, O) \tag{8}$$

The plan follows a structured workflow tailored to each category:

- *Single-hop*: The agent first attempts to `Extract` an entity. If successful, it executes a "scent-based" selection; otherwise, it falls back to a section-based strategy. Both paths then proceed to standard reasoning and generation, denoted as P_{std}.

$$P_{\text{s}} = \begin{cases} \texttt{Extract} \xrightarrow{\text{success}} \texttt{Select\_by\_Entity} \rightarrow P_{\text{std}} \\ \texttt{Extract} \xrightarrow{\text{fail}} \texttt{Select\_by\_Section} \rightarrow P_{\text{std}} \end{cases} \tag{9}$$

$$P_{\text{std}} = (\texttt{Graph} \parallel \texttt{Text}) \rightarrow \texttt{Skyline} \rightarrow \texttt{Reduce} \tag{10}$$

- *Multi-hop*: The agent first decomposes the problem, applies the Single-hop workflow P_{s} to each sub-problem, and finally synthesizes the results.

$$P_{\text{m}} = \texttt{Decompose} \rightarrow P_{\text{s}} \rightarrow \texttt{Map} \rightarrow \texttt{Reduce} \tag{11}$$

- *Global Aggregation*: The workflow involves applying a sequence of filters followed by synthesis.

$$P_{\text{global}} = \prod (\texttt{Filter\_Modal} \mid \texttt{Filter\_Range}) \rightarrow \texttt{Map} \rightarrow \texttt{Reduce} \tag{12}$$

Here, the symbol $\prod$ denotes the nested composition of filters, applying either a modal or range filter at each step.

5.3 Structured Execution

BookRAG executes the planned workflow P following an IFT-inspired "narrow-then-reason" strategy. Specifically, the `Selector` operators mirror the act of "navigating to information patches," narrowing the vast document space down to relevant scopes. Subsequently, the `Reasoner` operators perform "sense-making within patches," where they analyze and refine the information within these focused scopes. Finally, the `Synthesizer` generates the answer based on the processed evidence. This design minimizes the cost of attention by ensuring computational resources are focused solely on high-value data patches.

Scent/Filter-based Retrieval. The execution begins by narrowing the scope. Aligning with IFT, `Selector` operators identify relevant "patches" by following "information scents" (e.g., key entities in question) or applying explicit filter constraints. This process reduces the full node set N to a focused node subset N_s:

$$N_s = \texttt{Selector}(N, \text{params}_{\text{sel}}) \tag{13}$$

This pre-selection minimizes noise and ensures that subsequent reasoning is applied only to highly relevant contexts, optimizing the foraging cost. Subsequently, within this focused scope, `Reasoner` operators evaluate nodes using multiple dimensions, such as graph topology and semantic relevance. We then employ the `Skyline_Ranker` to obtain the final retrieval set. Unlike fixed top-k retrieval, the Skyline operator returns the *Pareto frontier* of nodes, retaining nodes that are valuable in at least one dimension while discarding dominated ones:

$$N_R = \texttt{Skyline\_Ranker}(\{S_G(n), S_T(n) \mid n \in N_s\}) \tag{14}$$

Analysis & Merging Generation. In the final stage, the `Synthesizer` operator generates the coherent answer by aggregating the refined evidence:

$$A = \texttt{Synthesizer}(q, N_R) \tag{15}$$

The `Map` operator performs fine-grained analysis on individual evidence blocks or sub-problems (from `Decompose`) to generate intermediate insights. The `Reduce` operator then aggregates these partial results, such as answers to decomposed sub-queries or statistical counts from a global filter, to construct the final response. This separation ensures that the system can handle both detailed content extraction and high-level reasoning synthesis effectively.

Figure 4(b) illustrates this workflow via a "single-hop" query. The agent first invokes (1) `Extract` the key entity (e.g., "car"), and then invokes (2) `Select` the relevant "Method" subtree, reducing the search space from 125 to 27 nodes. Within this narrowed scope, (3)-(4) it applies graph- and text-based `Reasoning` to score candidate nodes, followed by `Skyline` filtering to retain the most relevant nodes. Finally, (5) it invokes `Synthesizes` the selected nodes into the final answer. This trace illustrates how BookRAG dynamically composes operators to progressively narrow the search space, localize highly relevant evidence, and efficiently answer the query.

6 EXPERIMENTS

In our experiments, we evaluate BookRAG against several strong baseline methods, with an in-depth comparison of their efficiency and accuracy on document QA tasks.

6.1 Setup

Table 3: Datasets used in our experiments. EM and F1 denote Exact Match and F1-score, respectively.

Dataset	MMLongBench	M3DocVQA	Qasper
Questions	669	633	640
Documents	85	500	192
Avg. Pages	42.16	8.52	10.95
Avg. Images	25.92	3.51	3.43
Tokens	2,816,155	3,553,774	2,265,349
Metrics	EM, F1	EM, F1	Accuracy, F1

Datasets & Question Synthesis. We use three widely adopted benchmarking datasets for complex document QA tasks: MMLongBench [35], M3DocVQA [11], and Qasper [14]. MMLongBench is a comprehensive benchmark designed to evaluate QA capabilities on long-form documents, covering diverse categories such as guidebooks, financial reports, and industry documents. M3DocVQA is an open-domain benchmark designed to test RAG systems on a diverse collection of HTML documents sourced from Wikipedia pages[1]. Qasper is a QA dataset focused on scientific papers, where questions require retrieving evidence from the entire document. To support a fair comparison, we filter out documents that cannot be reliably handled by either the baselines or BookRAG, including those with severe visual degradation or lacking intrinsic logical hierarchies, such as presentation slides. Furthermore, to address the scarcity of Global Aggregation questions, which are nearly absent in M3DocVQA and Qasper, we synthesize 125 additional QA pairs per dataset using a human-in-the-loop pipeline. These additional pairs account for less than 20% of the final evaluation set and mainly supplement the evaluation of Global Aggregation questions. The statistics of the processed datasets are presented in Table 3, and more details regarding the filtering and synthesis processes are provided in the appendix of our technical report [57].

Metrics. We adhere to the official metrics specified by each dataset for QA. Our primary evaluation relies on Exact Match (EM), accuracy, and token-based F1-score. To assess efficiency, we measure: (1) *Online:* end-to-end time cost (from receiving queries to producing the final answer) and token usage (both prompt and completion tokens across all LLMs and VLMs); (2) *Offline:* total construction time, storage, and token consumption. Additionally, for methods including PDF parsing, we also evaluate retrieval recall. To establish the ground truth for this, we manually label the specific PDF blocks (e.g., texts, titles, tables, images, and formulas) required to answer each question. This labeling process is guided by the metadata of ground-truth evidence provided in each dataset; we filter candidate blocks using the given modality (all datasets), page numbers (MMLongBench), and evidence statements (Qasper). Any blocks that remained non-unique after this filtering process are manually annotated. In cases where a PDF parsing error made the ground-truth item unavailable, the retrieval recall for that query is recorded as 0.

Baselines. Our experiments consider three model configurations:

- **Conventional RAG:** These methods are the most common pipeline for document analysis, where the raw text is first extracted and then chunked into segments of a specified size. We select strong and widely used retrieval models: BM25 [44] and Vanilla RAG. We also implement Layout+Vanilla, a variant that uses document layout analysis for semantic chunking.
- **Graph-based RAG:** These methods first extract textual content from documents and then leverage graph data during retrieval. We select RAPTOR [45] and GraphRAG [17]. GraphRAG has two versions: GraphRAG-Global and GraphRAG-Local, which employ global and local search methods, respectively.
- **Layout segmented RAG:** This category encompasses methods that utilize layout analysis to segment document content into discrete structural units. We include: MM-Vanilla, which utilizes multimodal embeddings for visual and textual content; a tree-based method inspired by PageIndex [66], denoted as TreeTraverse, where an LLM navigates the document's tree structure; SuperRAG [63], which organizes diverse layout units into a graph based on their spatial proximity and reading order; DocETL [46], a declarative system for complex document processing; and GraphRanker, a graph-based method extended from HippoRAG [20] that applies Personalized PageRank [21] to rank the relevant nodes.

Implementation details. For a fair comparison, both BookRAG and all baseline methods are powered by a unified set of backbone models from the Qwen family. Specifically, we utilize Qwen3-8B [62] as the default LLM, Qwen2.5VL-30B [3] as the VLM, Qwen3-Embedding-0.6B [69] for text embedding, Qwen3-Reranker-4B [69] for reranking, and gme-Qwen2-VL-2B-Instruct [68] for multimodal embedding. All methods utilize the same high-quality parsing results from MinerU [52], unless a baseline explicitly requires a specialized parser, such as SuperRAG with Docling [32]. Methods based on fixed top-k retrieval use a unified setting with $k = 10$. For other baselines, we follow the settings recommended in their original papers or official implementations. All experiments are conducted serially on a high-performance server equipped with 8 NVIDIA RTX A5000 GPUs to eliminate resource contention and ensure a fair comparison. We set the gradient threshold g to 0.6, and more details are provided in the appendix of our technical report [57]. Our source code, prompts, and detailed configurations are available at github.com/sam234990/BookRAG.

6.2 Overall results

In this section, we present a comprehensive evaluation of BookRAG.

[1] https://www.wikipedia.org/

Table 4: Performance comparison of different methods across various datasets for solving complex document QA tasks. The best and second-best results are marked in bold and underlined, respectively.

Baseline Type	Method	MMLongBench		M3DocVQA		Qasper	
		(Exact Match)	(F1-score)	(Exact Match)	(F1-score)	(Accuracy)	(F1-score)
Conventional RAG	BM25	18.3	20.2	34.6	37.8	38.1	42.5
	Vanilla RAG	16.5	18.0	36.5	40.2	40.6	44.4
	Layout + Vanilla	18.1	19.8	36.9	40.2	40.7	44.6
Graph-based RAG	RAPTOR	21.3	21.8	34.3	37.3	39.4	44.1
	GraphRAG-Local	7.7	8.5	23.7	25.6	35.9	39.2
	GraphRAG-Global	5.3	5.6	20.2	22.0	24.0	24.1
Layout segmented RAG	MM-Vanilla	6.8	8.4	25.1	27.7	27.9	29.3
	Tree-Traverse	12.7	14.4	33.3	36.2	27.3	32.1
	GraphRanker	21.2	22.7	<u>43.0</u>	<u>47.8</u>	32.9	37.6
	SuperRAG	14.8	17.9	25.9	29.5	37.8	40.7
	DocETL	<u>27.5</u>	<u>28.6</u>	40.9	43.3	<u>42.3</u>	<u>50.4</u>
Our proposed	**BookRAG**	**43.8**	**44.9**	**61.0**	**66.2**	**55.2**	**61.1**

• **QA Performance of BookRAG**. We compare the QA performance of BookRAG against three categories of baselines, as shown in Table 4. The results indicate that BookRAG achieves state-of-the-art performance across all datasets, substantially outperforming the top-performing baseline by 18.0% in Exact Match on M3DocVQA. Layout + Vanilla consistently outperforms Vanilla RAG, confirming that layout parsing preserves essential structural information for better retrieval. Moreover, the suboptimal results of Tree-Traverse and GraphRanker highlight the limitations of relying solely on hierarchical navigation or graph-based reasoning, which often miss cross-sectional context or drift into irrelevant scopes. In contrast, BookRAG's superiority stems from the synergy of its unified Tree-Graph BookIndex and Agent-based Planning. By effectively classifying queries and configuring optimal workflows, our BookRAG overcomes limitations of context fragmentation and static query workflows within existing baselines, ensuring precise evidence retrieval and accurate generation.

Table 5: Retrieval recall comparison among layout-based methods. The best and second-best results are marked in bold and underlined, respectively.

Method	MMLongBench	M3DocVQA	Qasper
Layout + Vanilla	26.3	33.8	<u>33.5</u>
MM-Vanilla	7.5	19.7	14.9
Tree-Traverse	11.2	19.5	14.5
GraphRanker	<u>26.4</u>	<u>44.5</u>	28.6
BookRAG	**57.6**	**71.2**	**63.5**

• **Retrieval performance of BookRAG.** To validate our retrieval design, we evaluate the retrieval recall of BookRAG against other layout-based baselines on the ground-truth layout blocks. The experimental results demonstrate that BookRAG achieves the highest recall across all datasets, notably reaching 71.2% on M3DocVQA and significantly outperforming the next best baseline (GraphRanker, max 44.5%). This performance advantage stems from our IFT-inspired **Selector → Reasoner** workflow, where the `Selector` narrows the search to a precise *information patch* for the `Reasoner`'s analysis. Crucially, after the `Skyline_Ranker` process, the average number of retained nodes is 9.87, 6.86, and 8.6 across the three datasets, which is comparable to the standard top-k ($k = 10$) setting, ensuring high-quality retrieval without inflating the candidate size.

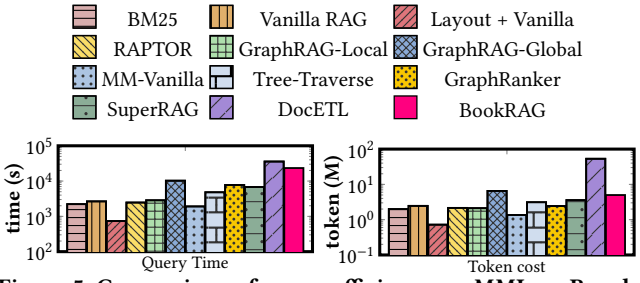


Figure 5: Comparison of query efficiency on MMLongBench. Bars follow the legend's left-to-right order.

• **Efficiency of BookRAG.** We further evaluate the efficiency in terms of query time and token consumption, as illustrated in Figure 5. Overall, BookRAG maintains time and token costs comparable to existing Graph-based RAG methods. While purely text-based RAG approaches generally exhibit lower latency and token usage due to the absence of VLM processing for images, BookRAG maintains a balanced efficiency among multimodal methods. In terms of token usage, BookRAG reduces consumption by an order of magnitude compared to the strongest baseline, DocETL. Notably, on the MMLongBench dataset, DocETL consumes over 53 million tokens, whereas BookRAG requires less than 5 million. Regarding query latency, our method also achieves a speedup of up to 2× compared to DocETL.

• **Index Construction and Storage.** We evaluate the offline overhead in terms of end-to-end construction time, storage size, and token consumption (Figures 6 and 7). Overall, BookRAG exhibits an offline indexing overhead comparable to existing graph-based methods; for instance, on the MMLongBench dataset, our construction time (145, 107s) is closely aligned with GraphRAG (172, 625s). In terms of storage, BookRAG maintains an efficient footprint with an average of 52.4 MB per document on MMLongBench, which is on par with the 43.3 MB of Layout + Vanilla. This confirms that our tree and graph structure contribute a negligible fraction to the total index size, which is otherwise dominated by high-dimensional

embeddings. While multimodal processing results in higher indexing costs (e.g., 36.9M for BookRAG vs. 15.5M for GraphRAG on Qasper), this one-time investment enables BookRAG to achieve superior efficiency and precision during online queries.

For brevity, Figures 5 and 6 focus on MMLongBench; remaining results for M3DocVQA and Qasper are provided in the appendix of our technical report [57]. The trends across all datasets remain consistent, further validating the efficiency of BookRAG.

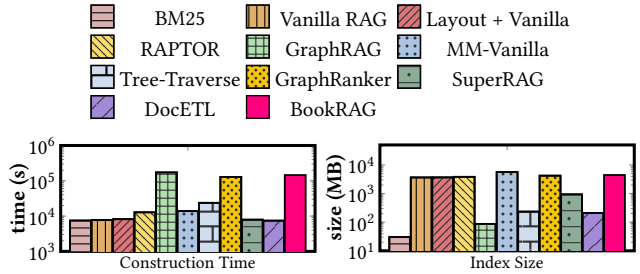


Figure 6: Comparison of Index Construction Overhead on MMLongBench. Bars follow the legend's left-to-right order.

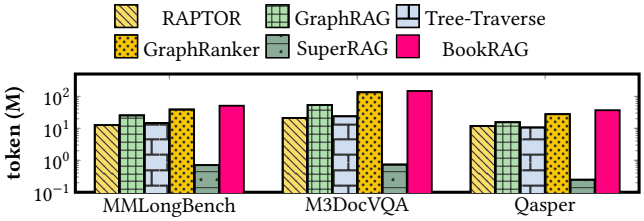


Figure 7: Token Consumption for Index Construction. Bars follow the legend's left-to-right order.

6.3 Detailed Analysis

In this section, we provide a more in-depth examination of our BookRAG.

• **Ablation study.** To evaluate the contribution of each core component to BookRAG, we design several variants by removing specific components:

- w/o Gradient ER: Replaces the gradient-based entity resolution with a Basic ER by merging the same-name entities.
- w/o Planning: Removes the Agent-based Planning, defaulting to a static, standard workflow for all queries, as per the plan for single-hop questions.
- w/o `Selector`: Removes the Selector operators, forcing Reasoners to score all candidate nodes.
- w/o `Graph_Reasoning`: Removes the `Graph_Reasoning` operator. Consequently, the `Skyline_Ranker` is also disabled as scoring becomes single-dimensional.
- w/o `Text_Reasoning`: Removes the `Text_Reasoning` operator. Similarly, the `Skyline_Ranker` is disabled, relying solely on graph-based scores.
- w/o `Map`: Removes the `Map` operator and adds raw retrieved contexts directly to the final synthesis (`Reduce` operator) until the LLM's token limit is reached.
- w/o `Plan & Ops`: Retrieves the top-k most relevant nodes and entities from the tree and graph, respectively, and feeds the aggregated information to the LLM for generation.

Table 6: Comparing the QA performance of different variants of BookRAG. EM and F1 denote Exact Match and F1-score, respectively.

Method variants	MMLongBench		Qasper	
	EM	F1	Accuracy	F1
BookRAG (Full)	43.8	44.9	55.2	61.1
w/o gradient ER	40.1	42.8	48.9	57.3
w/o Planning	30.8	33.2	40.9	48.5
w/o `Selector`	42.5	43.1	52.5	59.1
w/o `Graph_Reasoning`	39.8	41.5	51.4	58.4
w/o `Text_Reasoning`	39.0	40.3	47.2	52.5
w/o `Map`	40.8	42.9	46.1	54.9
w/o `Plan & Ops`	21.1	22.1	41.3	47.2

The first three variants evaluate KG quality, Agent-based Planning, and IFT-inspired selection, respectively. The remaining variants test the contribution of individual operators and the overall planning-and-operator workflow beyond simple retrieval and direct generation. As shown in Table 6, the performance degradation across all variants confirms the essential role of each module in BookRAG. Specifically, the performance drop in the *w/o Gradient ER* variant highlights the critical role of a high-quality, connectivity-rich KG in supporting effective reasoning. Removing the *Planning* mechanism results in the most significant performance loss, confirming that a static workflow is insufficient for handling diverse types of queries. The *w/o Selector* variant maintains comparable accuracy but incurs much higher cost; for example, on Qasper, the token usage exceeds 2×, while the average number of retrieved nodes increases from 8.6 to 15.9. Without the structural constraints of the `Selector`, the reasoning operators have to explore a much larger candidate space in the BookIndex. This result highlights the importance of the `Selector` for improving cost-efficiency by narrowing the search space before reasoning. The performance declines in w/o `Map` and w/o `Plan & Ops` further confirm the value of coordinated planning and specialized operators for preserving long-context coherence, relative to a vanilla RAG pipeline.

• **Impact of Gradient-based Entity Resolution.** To evaluate the quality of our constructed KG, we compare the graph statistics of our Gradient-based ER against a Basic KG construction. The Basic setting employs simple exact name matching for entity merging, which is standard practice in many graph-based methods. Figure 9 presents the comparative results, normalizing the metrics (Entity count, Density, Diameter of the Largest Connected Component, and Number of Connected Components) against the Basic baseline. The results demonstrate that our Gradient-based ER significantly improves KG quality. Specifically, it reduces the number of entities (by 12%) while substantially boosting graph density (by over 20% across datasets). This structural shift indicates that our ER module effectively identifies the same concepts expressed under different names. Consequently, the resulting graphs are more compact and cohesive, as evidenced by the reduced diameter and fewer connected components, which mitigates graph fragmentation and facilitates better connectivity for graph reasoning.

• **Sensitivity Analysis of g.** We further investigate the sensitivity of the gradient threshold g by varying it from 0.2 to 0.8 across different reranker backbones (Qwen3-Reranker-4B and bge-reranker-v2-m3 [8]). As illustrated in Figure 10, the graph statistics remain remarkably stable across this entire range. This stability

BookRAG response to a multi-hop case from Qasper

Question: What is the difference in performance between the interpretable system (e.g., vectors and cosine distance) and LSTM with ELMo system?
Human-written answer: Accuracy of best interpretable system was 0.3945 while accuracy of LSTM-ELMo net was 0.6818.
Evidence 1: The experimental results are presented in Table 1. Diacritic swapping showed a remarkably poor performance </>
Evidence 2: Test results for all the methods used. The loss measure is cross-entropy </>

Agent-based Planning: This is a multi-hop query. Here is the Decompose operator:

```
Sub_question 1: "What is the performance metric of the interpretable system (e.g., vectors and cosine distance)?"
Sub_question 2: "What is the performance metric of the LSTM with ELMo system?"
```

Process: To resolve the multi-hop dependency, BookRAG orchestrates parallel single-hop pipelines (comprising `Select`, `Reason`, and `Synthesizer` operators) to navigate disjoint document regions and localize each sub-question's evidence:

Sub-q 1 Execution: Identifies entities (*e.g.*, "cosine distance"), selects sub-trees under "5. Results", and prunes the search space from 53 to 13 nodes. Subsequent `Reasoners` retrieve 7 evidence nodes.

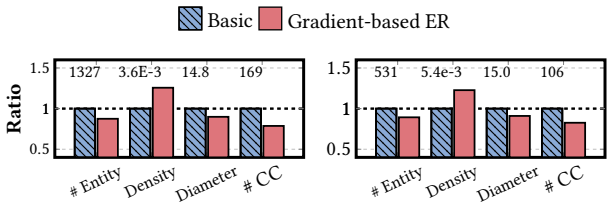
Sub-q 2 Execution: Identifies architecture entities (*e.g.*, "LSTM-ELMo"), targets "4. Experimental setup", and prunes the space from 53 to 7 nodes, retrieving 5 critical nodes.

Intermediate Synthesis: Partial responses are generated based on localized evidence:

```
Answer 1: "Summary:\n- Performance Metric: Accuracy (0.3945) and </>"
Answer 2: "The performance metric of the LSTM with ELMo system (LSTM-ELMo net) is as follows:\n Accuracy: 0.6818 </>"
```

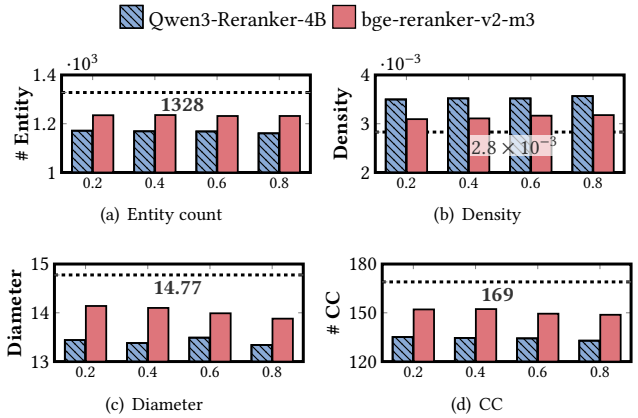
Output: The interpretable system (using vector distance and cosine distance) **has an accuracy of 0.3945**, while the LSTM with ELMo system **has an accuracy of 0.6818**. The difference in accuracy between the two systems is 0.6818 - 0.3945 = 0.2873. </>

Figure 8: Case study of a response to a multi-hop question from Qasper. For conciseness, only high-level orchestration steps are shown, while nested retrieval operators for each sub-query are omitted. CYAN TEXT highlights correct content generated by BookRAG. GRAY TEXT describes the internal process, and </> marks omitted irrelevant parts.



(a) MMLongBench (b) Qasper

Figure 9: Comparison of graph statistics. Values are normalized to the Basic setting (Blue bars with diagonal lines, Baseline=1.0). Absolute values for Basic are annotated. Note that density values are abbreviated (e.g., 3.6E-3 denotes 3.6×10^{-3}).



(a) Entity count (b) Density

(c) Diameter (d) CC

Figure 10: Sensitivity analysis of graph statistics by varying gradient thresholds g and different rerankers. The horizontal dashed lines represent the metrics of the Basic KG construction (the values are annotated on the plots).

confirms that the isolation of high-relevance entities is determined by the intrinsic scoring distributions of the rerankers rather than by specific threshold settings. While more powerful rerankers (Qwen3-Reranker-4B) consistently yield more cohesive graphs than smaller ones, the overall graph quality remains robust to the choice of g, demonstrating that our ER framework does not require extensive document-specific tuning.

• **Case Study.** Figure 8 illustrates BookRAG's execution workflow for a complex multi-hop query. The process demonstrates that BookRAG correctly decomposes a reasoning question into two subtasks, each of which internally triggers a full retrieval pipeline to navigate distinct document regions. This indicates that scattered evidence is accurately synthesized while nested operators effectively prune the search space for each sub-problem.

We provide additional detailed experiments in the appendix of our technical report [57], including a comprehensive *Error Response Analysis*, a performance breakdown across *Query Categories*, and an extended *Case Study* with a more comprehensive set of examples. These evaluations further reveal characteristic failure patterns, highlight the importance of query categorization for plan construction, and illustrate the workflow of BookRAG.

7 CONCLUSION

In this paper, we propose BookRAG, a novel method built upon BookIndex, a document-native, structured Tree-Graph index specifically designed to capture intricate relations within structured documents. By employing an agent-based method to dynamically configure retrieval and reasoning operators, our approach achieves state-of-the-art performance on multiple benchmarks, demonstrating significant superiority over existing baselines in both retrieval precision and answer accuracy. In the future, we will explore an integrated document-native database system that supports data formatting, knowledge extraction, and intelligent querying.

ACKNOWLEDGMENTS

This work was supported by the 1+1+1 CUHK-CUHK(SZ)-GDSTC Joint Collaboration Fund under Grant 2025A0505000045.

REFERENCES

[1] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. 2023. Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes. *Proceedings of the VLDB Endowment* 17, 2 (2023), 92–105.

[2] Akari Asai, Zeqiu Wu, Yizhong Wang, et al. 2024. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. In *International Conference on Learning Representations (ICLR)*.

[3] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. 2025. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923* (2025).

[4] Camille Barboule, Benjamin Piwowarski, and Yoan Chabot. 2025. Survey on Question Answering over Visually Rich Documents: Methods, Challenges, and Trends. *arXiv preprint arXiv:2501.02235* (2025).

[5] Yukun Cao, Zengyi Gao, Zhiyang Li, Xike Xie, S. Kevin Zhou, and Jianliang Xu. 2025. LEGO-GraphRAG: Modularizing Graph-Based Retrieval-Augmented Generation for Design Space Exploration. *Proc. VLDB Endow.* 18, 10 (June 2025), 3269–3283. https://doi.org/10.14778/3748191.3748194

[6] Chengliang Chai, Jiajun Li, Yuhao Deng, Yuanhao Zhong, Ye Yuan, Guoren Wang, and Lei Cao. 2025. Doctopus: Budget-aware structural table extraction from unstructured documents. *Proceedings of the VLDB Endowment* 18, 11 (2025), 3695–3707.

[7] Ilias Chalkidis, Manos Fergadiotis, Prodromos Malakasiotis, Nikolaos Aletras, and Ion Androutsopoulos. 2020. LEGAL-BERT: The muppets straight out of law school. *arXiv preprint arXiv:2010.02559* (2020).

[8] Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In *Findings of the association for computational linguistics: ACL 2024*. 2318–2335.

[9] Sibei Chen, Yeye He, Weiwei Cui, Ju Fan, Song Ge, Haidong Zhang, Dongmei Zhang, and Surajit Chaudhuri. 2024. Auto-Formula: Recommend Formulas in Spreadsheets using Contrastive Learning for Table Representations. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–27.

[10] Sibei Chen, Nan Tang, Ju Fan, Xuemi Yan, Chengliang Chai, Guoliang Li, and Xiaoyong Du. 2023. Haipipe: Combining human-generated and machine-generated pipelines for data preparation. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–26.

[11] Jaemin Cho, Debanjan Mahata, Ozan Irsoy, Yujie He, and Mohit Bansal. 2024. M3docrag: Multi-modal retrieval is what you need for multi-page multi-document understanding. *arXiv preprint arXiv:2411.04952* (2024).

[12] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. 2020. An overview of end-to-end entity resolution for big data. *ACM Computing Surveys (CSUR)* 53, 6 (2020), 1–42.

[13] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261* (2025).

[14] Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A Smith, and Matt Gardner. 2021. A dataset of information-seeking questions and answers anchored in research papers. *arXiv preprint arXiv:2105.03011* (2021).

[15] Xavier Daull, Patrice Bellot, Emmanuel Bruno, Vincent Martin, and Elisabeth Murisasco. 2023. Complex QA and language models hybrid architectures, Survey. *arXiv preprint arXiv:2302.09051* (2023).

[16] Mostafa Dehghani, Basil Mustafa, Josip Djolonga, Jonathan Heek, Matthias Minderer, Mathilde Caron, Andreas Steiner, Joan Puigcerver, Robert Geirhos, Ibrahim M. Alabdulmohsin, Avital Oliver, Piotr Padlewski, Alexey A. Gritsenko, Mario Lucic, and Neil Houlsby. 2023. Patch n' Pack: NaViT, a Vision Transformer for any Aspect Ratio and Resolution. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.). Retrieved May 19, 2026 from http://papers.nips.cc/paper_files/paper/2023/hash/06ea400b9b7cfce6428ec27a371632eb-Abstract-Conference.html

[17] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130* (2024).

[18] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* (2023).

[19] Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. 2024. LightRAG: Simple and Fast Retrieval-Augmented Generation. *arXiv e-prints* (2024), arXiv–2410.

[20] Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. HippoRAG: Neurobiologically Inspired Long-Term Memory for Large Language Models. *arXiv preprint arXiv:2405.14831* (2024).

[21] Taher H Haveliwala. 2002. Topic-sensitive pagerank. In *Proceedings of the 11th international conference on World Wide Web*. 517–526.

[22] Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh V Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. 2024. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering. *arXiv preprint arXiv:2402.07630* (2024).

[23] Yucheng Hu and Yuxing Lu. 2024. Rag and rau: A survey on retrieval-augmented language model in natural language processing. *arXiv preprint arXiv:2404.19543* (2024).

[24] Soyeong Jeong, Jinheon Baek, et al. 2024. Adaptive-RAG: Learning to Adapt Retrieval-Augmented Large Language Models through Question Complexity. *arXiv preprint arXiv:2403.14403* (2024).

[25] Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C Park. 2024. Adaptive-rag: Learning to adapt retrieval-augmented large language models through question complexity. *arXiv preprint arXiv:2403.14403* (2024).

[26] Tengjun Jin, Yuxuan Zhu, and Daniel Kang. 2025. ELT-Bench: An End-to-End Benchmark for Evaluating AI Agents on ELT Pipelines. *arXiv preprint arXiv:2504.04808* (2025).

[27] Geewook Kim, Teakgyu Hong, Moonbin Yim, JeongYeon Nam, Jinyoung Park, Jinyeong Yim, Wonseok Hwang, Sangdoo Yun, Dongyoon Han, and Seunghyun Park. 2022. Ocr-free document understanding transformer. In *European Conference on Computer Vision*. Springer, 498–517.

[28] Dawei Li, Shu Yang, Zhen Tan, Jae Young Baik, Sukwon Yun, Joseph Lee, Aaron Chacko, Bojian Hou, Duy Duong-Tran, Ying Ding, et al. 2024. DALK: Dynamic Co-Augmentation of LLMs and KG to answer Alzheimer's Disease Questions with Scientific Literature. *arXiv preprint arXiv:2405.04819* (2024).

[29] Guoliang Li, Jiayi Wang, Chenyang Zhang, and Jiannan Wang. 2025. Data+ AI: LLM4Data and Data4LLM. In *Companion of the 2025 International Conference on Management of Data*. 837–843.

[30] Yinheng Li, Shaofei Wang, Han Ding, and Hang Chen. 2023. Large language models in finance: A survey. In *Proceedings of the fourth ACM international conference on AI in finance*. 374–382.

[31] Zhaodonghui Li, Haitao Yuan, Huiming Wang, Gao Cong, and Lidong Bing. 2025. LLM-R2: A Large Language Model Enhanced Rule-based Rewrite System for Boosting Query Efficiency. *Proceedings of the VLDB Endowment* 1, 18 (2025), 53–65.

[32] Nikolaos Livathinos, Christoph Auer, Maksym Lysak, Ahmed Nassar, Michele Dolfi, Panos Vagenas, Cesar Berrospi Ramis, Matteo Omenetti, Kasper Dinkla, Yusik Kim, et al. 2025. Docling: An efficient open-source toolkit for ai-driven document conversion. *arXiv preprint arXiv:2501.17887* (2025).

[33] Haoyu Lu, Wen Liu, Bo Zhang, et al. 2024. DeepSeek-VL: Towards Real-World Vision-Language Understanding. *arXiv preprint arXiv:2403.05525* (2024).

[34] Shengjie Ma, Chengjin Xu, Xuhui Jiang, Muzhi Li, Huaren Qu, Cehao Yang, Jiaxin Mao, and Jian Guo. 2024. Think-on-Graph 2.0: Deep and Faithful Large Language Model Reasoning with Knowledge-guided Retrieval Augmented Generation. *arXiv preprint arXiv:2407.10805* (2024).

[35] Yubo Ma, Yuhang Zang, Liangyu Chen, Meiqi Chen, Yizhu Jiao, Xinze Li, Xinyuan Lu, Ziyu Liu, Yan Ma, Xiaoyi Dong, et al. 2024. Mmlongbench-doc: Benchmarking long-context document understanding with visualizations. *Advances in Neural Information Processing Systems* 37 (2024), 95963–96010.

[36] Zan Ahmad Naeem, Mohammad Shahmeer Ahmad, Mohamed Eltabakh, Mourad Ouzzani, and Nan Tang. 2024. RetClean: Retrieval-Based Data Cleaning Using LLMs and Data Lakes. *Proceedings of the VLDB Endowment* 17, 12 (2024), 4421–4424.

[37] Avanika Narayan, Ines Chami, Laurel Orr, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? *Proceedings of the VLDB Endowment* 16, 4 (2022), 738–746.

[38] Yuqi Nie, Yaxuan Kong, Xiaowen Dong, John M Mulvey, H Vincent Poor, Qingsong Wen, and Stefan Zohren. 2024. A Survey of Large Language Models for Financial Applications: Progress, Prospects and Challenges. *arXiv preprint arXiv:2406.11903* (2024).

[39] Arash Dargahi Nobari and Davood Rafiei. 2024. TabulaX: Leveraging Large Language Models for Multi-Class Table Transformations. *arXiv preprint arXiv:2411.17110* (2024).

[40] Liana Patel, Siddharth Jha, Melissa Pan, Harshit Gupta, Parth Asawa, Carlos Guestrin, and Matei Zaharia. 2025. Semantic Operators and Their Optimization: Enabling LLM-Based Data Processing with Accuracy Guarantees in LOTUS. *Proceedings of the VLDB Endowment* 18, 11 (2025), 4171–4184.

[41] Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. 2024. Graph retrieval-augmented generation: A survey. *arXiv preprint arXiv:2408.08921* (2024).

[42] Peter Pirolli and Stuart Card. 1995. Information foraging in information access environments. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 51–58.

[43] Yichen Qian, Yongyi He, Rong Zhu, Jintao Huang, Zhijian Ma, Haibin Wang, Yaohua Wang, Xiuyu Sun, Defu Lian, Bolin Ding, et al. 2024. UniDM: A Unified Framework for Data Manipulation with Large Language Models. *Proceedings of Machine Learning and Systems* 6 (2024), 465–482.

[44] Stephen E Robertson and Steve Walker. 1994. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR'94: Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, organised by Dublin City University*. Springer, 232–241.

[45] Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D Manning. 2024. Raptor: Recursive abstractive processing for tree-organized retrieval. *arXiv preprint arXiv:2401.18059* (2024).

[46] Shreya Shankar, Tristan Chambers, Tarak Shah, Aditya G Parameswaran, and Eugene Wu. 2024. Docetl: Agentic query rewriting and evaluation for complex document processing. *arXiv preprint arXiv:2410.12189* (2024).

[47] Shamane Siriwardhana, Rivindu Weerasekera, Elliott Wen, Tharindu Kaluarachchi, Rajib Rana, and Suranga Nanayakkara. 2023. Improving the domain adaptation of retrieval augmented generation (RAG) models for open domain question answering. *Transactions of the Association for Computational Linguistics* 11 (2023), 1–17.

[48] Solutions Review Editors. 2019. 80 Percent of Your Data Will Be Unstructured in Five Years. https://solutionsreview.com/data-management/80-percent-of-your-data-will-be-unstructured-in-five-years/. Accessed: 2023-10-27.

[49] Yaodong Su, Yixiang Fang, Yingli Zhou, Quanqing Xu, and Chuanhui Yang. 2025. Clue-RAG: Towards Accurate and Cost-Efficient Graph-based RAG via Multi-Partite Graph and Query-Driven Iterative Retrieval. *arXiv preprint arXiv:2507.08445* (2025).

[50] Zhaoyan Sun, Xuanhe Zhou, and Guoliang Li. 2024. R-Bot: An LLM-based Query Rewrite System. *arXiv preprint arXiv:2412.01661* (2024).

[51] Vincent A Traag, Ludo Waltman, and Nees Jan Van Eck. 2019. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific reports* 9, 1 (2019), 1–12.

[52] Bin Wang, Chao Xu, Xiaomeng Zhao, Linke Ouyang, Fan Wu, Zhiyuan Zhao, Rui Xu, Kaiwen Liu, Yuan Qu, Fukai Shang, et al. 2024. Mineru: An open-source solution for precise document content extraction. *arXiv preprint arXiv:2409.18839* (2024).

[53] Jiayi Wang and Guoliang Li. 2025. Aop: Automated and interactive llm pipeline orchestration for answering complex queries. CIDR.

[54] Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. 2024. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191* (2024).

[55] Shu Wang, Yixiang Fang, Yingli Zhou, Xilin Liu, and Yuchi Ma. 2025. ArchRAG: Attributed Community-based Hierarchical Retrieval-Augmented Generation. *arXiv preprint arXiv:2502.09891* (2025).

[56] Shen Wang, Tianlong Xu, Hang Li, Chaoli Zhang, Joleen Liang, Jiliang Tang, Philip S Yu, and Qingsong Wen. 2024. Large language models for education: A survey and outlook. *arXiv preprint arXiv:2403.18105* (2024).

[57] Shu Wang, Yingli Zhou, and Yixiang Fang. 2026. BookRAG: A Hierarchical Structure-aware Index-based Approach for Complex Document Question Answering. Retrieved May 19, 2026 from https://github.com/sam234990/BookRAG

[58] Yubo Wang, Haoyang Li, Fei Teng, and Lei Chen. 2025. AGRAG: Advanced Graph-based Retrieval-Augmented Generation for LLMs. *CoRR* abs/2511.05549 (2025). https://doi.org/10.48550/ARXIV.2511.05549 arXiv:2511.05549

[59] Yubo Wang, Haoyang Li, Fei Teng, and Lei Chen. 2026. GORAG: Graph-based Online Retrieval Augmented Generation for Dynamic Few-shot Social Media Text Classification. In *Proceedings of the ACM Web Conference 2026, WWW 2026, Dubai, United Arab Emirates, originally scheduled for April 13-17, 2026, rescheduled for June 29 - July 3, 2026*, Hakim Hacid, Yoelle Maarek, Francesco Bonchi, Ido Guy, and Emine Yilmaz (Eds.). ACM, 9148–9159. https://doi.org/10.1145/3774904.3793011

[60] Yu Wang, Nedim Lipka, Ryan A Rossi, Alexa Siu, Ruiyi Zhang, and Tyler Derr. 2024. Knowledge graph prompting for multi-document question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 19206–19214.

[61] Shi-Qi Yan, Jia-Chen Gu, Yun Zhu, and Zhen-Hua Ling. 2024. Corrective Retrieval Augmented Generation. *arXiv preprint arXiv:2401.15884* (2024).

[62] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388* (2025).

[63] Chening Yang, Duy-Khanh Vu, Minh-Tien Nguyen, Xuan-Quang Nguyen, Linh Nguyen, and Hung Le. 2025. SuperRAG: Beyond RAG with Layout-Aware Graph Modeling. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: Industry Track)*. 544–557.

[64] Murong Yue. 2025. A survey of large language model agents for question answering. *arXiv preprint arXiv:2503.19213* (2025).

[65] Fangyuan Zhang, Zhengjun Huang, Yingli Zhou, Qintian Guo, Zhixun Li, Wensheng Luo, Di Jiang, Yixiang Fang, and Xiaofang Zhou. 2025. Erarag: Efficient and incremental retrieval augmented generation for growing corpora. *arXiv preprint arXiv:2506.20963* (2025).

[66] Mingtian Zhang, Yu Tang, and PageIndex Team. 2025. PageIndex: Next-Generation Vectorless, Reasoning-based RAG. *PageIndex Blog* (September 2025). https://pageindex.ai/blog/pageindex-intro.

[67] Qinggang Zhang, Shengyuan Chen, Yuanchen Bei, Zheng Yuan, Huachi Zhou, Zijin Hong, Hao Chen, Yilin Xiao, Chuang Zhou, Junnan Dong, et al. 2025. A survey of graph retrieval-augmented generation for customized large language models. *arXiv preprint arXiv:2501.13958* (2025).

[68] Xin Zhang, Yanzhao Zhang, Wen Xie, Mingxin Li, Ziqi Dai, Dingkun Long, Pengjun Xie, Meishan Zhang, Wenjie Li, and Min Zhang. 2024. GME: Improving Universal Multimodal Retrieval by Multimodal LLMs. *arXiv preprint arXiv:2412.16855* (2024).

[69] Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, et al. 2025. Qwen3 Embedding: Advancing Text Embedding and Reranking Through Foundation Models. *arXiv preprint arXiv:2506.05176* (2025).

[70] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223* 1, 2 (2023).

[71] Yingli Zhou, Yaodong Su, Youran Sun, Shu Wang, Taotao Wang, Runyuan He, Yongwei Zhang, Sicong Liang, Xilin Liu, Yuchi Ma, et al. 2025. In-depth Analysis of Graph-based RAG in a Unified Framework. *arXiv preprint arXiv:2503.04338* (2025).

[72] Yutao Zhu, Huaying Yuan, Shuting Wang, Jiongnan Liu, Wenhan Liu, Chenlong Deng, Haonan Chen, Zheng Liu, Zhicheng Dou, and Ji-Rong Wen. 2023. Large language models for information retrieval: A survey. *ACM Transactions on Information Systems* (2023).